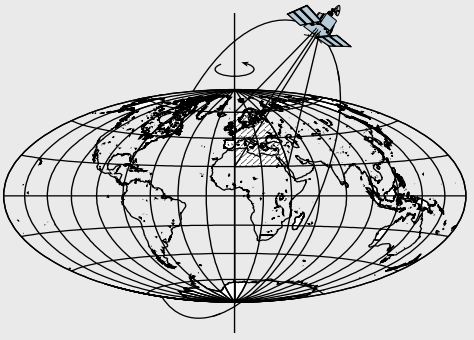
The Ohio State University Stackfiles for Satellite Radar Altimeter Data

by

Yuchan Yi



Report No. 495

(from an original report, dated 2000)

Geodetic Science

The Ohio State University
Columbus, Ohio 43210

May 2010

THE OHIO STATE UNIVERSITY STACKFILES FOR SATELLITE RADAR ALTIMETER DATA

BY

YUCHAN YI

Report No. 495

(from an original report, dated 2000)

Geodetic Science

School of Earth Sciences

The Ohio State University

Columbus, Ohio 43210

May 2010

ABSTRACT

This document describes the OSU stackfile database for satellite radar altimetry and software that is used to access and maintain the database system. The stackfile database system can be viewed as a reformatted version of Geophysical Data Record (GDR) data products of satellite radar altimeters. A stackfile database is accessible using 2-dimensional location indices of nominal ground tracks while the GDR products are registered in time along actual ground tracks. The third dimension of a stackfile is the repeat cycle of a satellite altimeter mission. The purpose of this document is to use it as a user's guide of the OSU stackfile databases installed on a unix/linux server.

ACKNOWLEDGMENTS

The original version of stackfiles was designed by Gerhard L.H. Kruizinga, Center for Space Research, University of Texas at Austin, 1994. Most part of the text in this document was borrowed from the CSR technical memo "The New Stackfiles" originally written by G.L.H. Kruizinga in the summer of 1994 and updated on August 21, 1998.

Table of Contents

0 Updates over a 1998 CSR Stackfile

These are features that are additional to the 1998 version of the Center for Space Research (CSR) stackfile system (The New Stackfiles, 1998; Kruizinga, 1997):

a. The data type of slot latitude and longitude arrays has been changed to Double Precision (REAL *8) from Single Precision (REAL *4). The arrays of slot latitude/longitude are accessed through routine sfgetlatslongs.

b. The first and last repeating cycle numbers of data available in the stackfiles are returned from routine sfgetsize.

c. Unit of Significant Wave Height (SWH) has been changed to cm from dm and that of Sigma-naught to 0.01 dB from 0.1 dB.

d. 10 Hz - 1 Hz Sea Surface Height (SSH) residuals are accessible through routine sfgetdssh10.

1 Introduction

1.1 Altimeter Satellites

The ground tracks of altimeter satellites Topex/Poseidon, ERS-1/2, and Geosat/Exact Repeat Mission (ERM) repeat after certain duration of nodal days. Here is a summary of pertinent orbital characteristics. Nodal days and revolutions refer to one complete cycle.

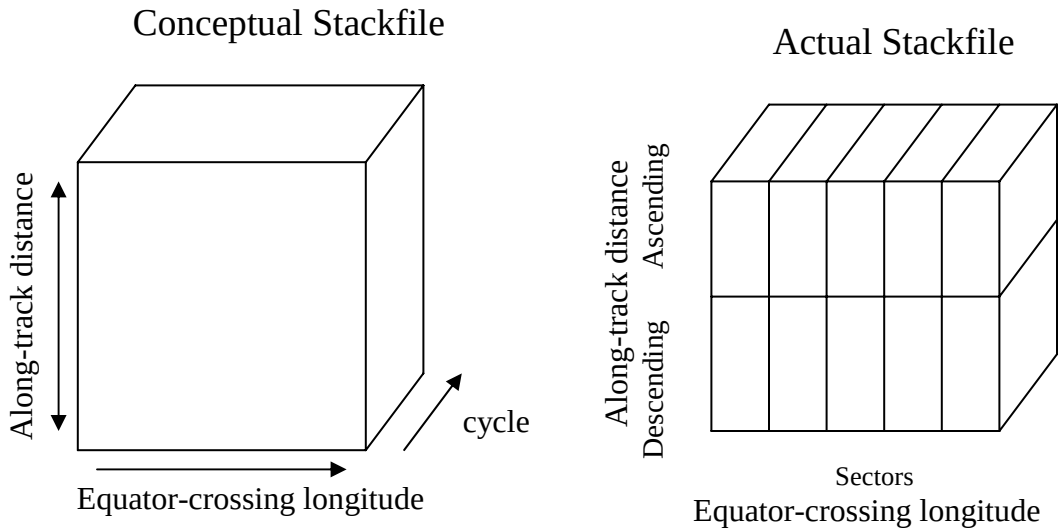
	Topex/Poseidon	ERS-1 Phases C/G ERS-2	Geosat ERM
Orbital period	6745.8 sec	6035.9 sec	6037.6 sec
Inclination	66.03°	98.58°	108.06°
Nodal days	9.9	35	17
Revolutions	127	501	244

The height of ocean surface varies with time due to changing currents and density of the ocean. Repeating ground tracks mean that each satellite revisits the same areas every repeat cycle, allowing development of time series of ocean variability.

A radar altimeter on each of the satellites continuously measures the distance between the satellite and ocean surface. Researchers receive the altimeter measurements as the Geophysical Data Record (GDR). Each GDR summarizes approximately 1 second of radar altimeter measurement, or more than 31 million GDRs per satellite a year. Each GDR contains a time tag, latitude and longitude of the satellite, and a short series of ocean surface height measurements made during the one second interval. GDRs also contain a number of other items such as environmental corrections and modeled effects.

1.2 Stackfiles

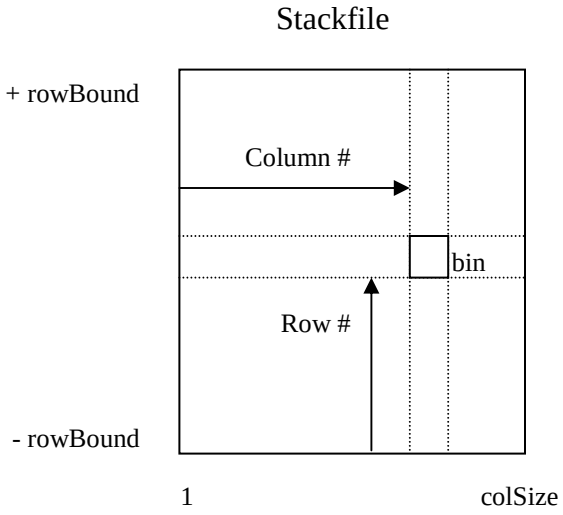
The Center for Space Research (CSR) at The University of Texas at Austin condenses GDRs into what are called "stackfiles." Each stackfile can be viewed as a three-dimensional array. The three dimensions represent (1) the distance from the equator along an orbit (row number); (2) which orbit of a repeat cycle (column number, actually, an equator-crossing longitude); and (3) each of the repeat cycles.



In an actual implementation, this monolith is cleaved along two of the dimensions. One cutting plane divides the set of along-track distance values. Each complete orbit is separated into ascending and descending passes (halves of orbit). Ascending passes begin at the southern extreme latitude and extend to the northern extreme; descending passes cover the other part of complete orbits.

The other kind of cutting planes were devised to keep the size of files manageable. They occur in the orbit dimension, sectoring the earth in effect. They separate the stackfile into what are called sector files. In current version of stackfiles, GDR data have been preprocessed and edited before storing them into stackfiles. Users may want to apply further data editing according to their own stringent edit criteria to use high quality data only out of stackfiles.

Data are stored into bins approximately 1 second of time long along the satellite ground tracks. A bin contains all the information from various repeat cycles measured over a particular area of the earth. In stackfiles, bins are addressed with row and column indices. Each column corresponds to a particular pass or orbit. Columns have been numbered in the order of equtor-crossing longitude not in the time order. Each row corresponds to a particular latitude.



Rows are numbered in such a way that the bin closest to the equator is row 0; negative row indices are south of the equator and positive ones north. Columns are indexed from 1 to the total number of equator crossings, increasing eastward. Likewise, repeat cycles or slots are indexed from 1 to the total number of repeat cycles.

Each bin is composed of a header and a number of slots, one per cycle. The bin header contains the bin center's latitude and longitude, number of slots with valid height value, mean and standard deviation of these heights, height bias, ocean depth (bathymetry), and eight logical flags. Presently only two of these flags are defined to indicate whether the bin center is over the ocean or land, etc.

To save space in the file, individual slot heights are stored as displacements from a height bias, which is representative of the entire bin. This height bias is stored in the bin header. The height bias is calculated from a geoid model. No sea surface height data over in-land waters are included in current stackfiles because of possibly large discrepancy between water surface height and local geoid surface.

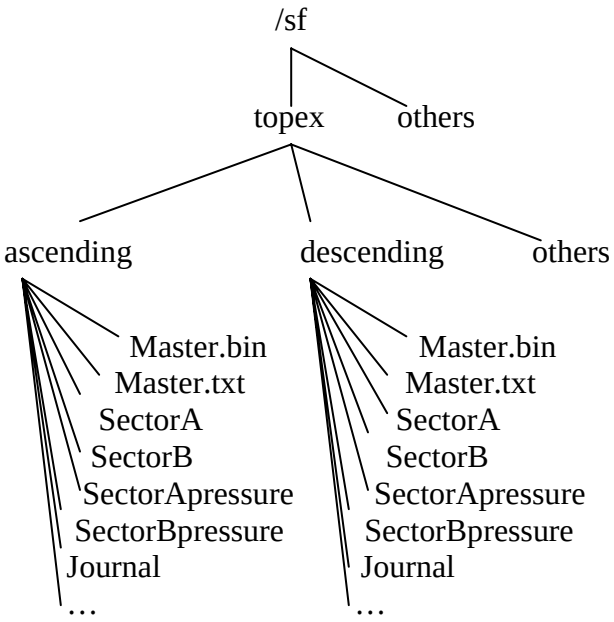
Slots of each bin contain the ocean surface height measured during corresponding repeat cycles (actually, the height displacement from bias is packed in files), eight logical flags, latitude and longitude displacements from the bin center, and other geophysical and environmental data such as the surface atmospheric pressure. The unit of height displacement is mm and those of latitude and longitude displacements are micro-degrees.

Slot flags indicate whether or not height is missing, displacements out of range, surface is covered with ice, and certain corrections have been applied to height data.

Each of geophysical and environmental data is stored in an independent stackfile separated from each other and from the height stackfile, which contains bin header and geographic lat/long information as described earlier. Currently, these are for surface pressure, ionospheric and wet troposheric corrections, electromagnetic (EM) bias correction, ocean tide correction, sigma-0 and significant wave height (SWH) of the ocean surface, and 10 Hz -1 Hz sea surface height (SSH) displacements.

In addition to the stackfile itself, there is a "Master.txt" file where useful information can be stored about the satellite and stackfile. This information can be retrieved by users' programs and used. Master.txt is an ASCII file. A scheme used at JPL was adopted where entries are organized group-wise. For example, there is a group named IDENTIFICATION. Within this group there are two items, SATELLITE-NAME and SATELLITE-ID. The value of item SATELLITE-NAME is a character string of the satellite name such as Topex/Poseidon. To get this information, a user's program supplies the group and item identifiers and a subroutine retrieves the corresponding text value from the stackfile's Master.txt file.

2 Implementation



A stackfile is a collection of data files stored in a directory. The directory is the stackfile's "name." A direction (ascending or descending) of each satellite has its own directory. In the figure, there are two stackfiles which are named /sf/topex/ascending and /sf/topex/descending.

Stackfiles are sectored in order to keep the size of each file below about 100 MB. Conceptually, a single stackfile is cleaved vertically so that the first set of columns are in a sector file, the next set of columns in the next sector file, and so on.

The files in the stackfile directory have standard names. One standard file is a binary file named Master.bin which contains the file dimensions (# rows, # columns, # slots), the total number of sector files, etc. The various sector files which store the actual height data also have standard names, SectorA, SectorB, etc. Similar files store other geophysical and environmental data such as the surface atmospheric pressure.

Each directory contains another master file in addition to the one described above. This Master.txt file contains textual information on satellite and stackfile. Finally, each directory contains a journal file where a log or journal is kept for every access of user programs.

2.1 Multiple Stackfiles

It is possible for a program to open multiple stackfiles simultaneously, for example, both the ascending and descending stackfiles, or stackfiles of distinct satellites. To implement this feature, when the stackfile is first opened, a "stackfile number" is returned which is similar to the logical unit number of a file. The stackfile number must be provided to

subsequent routines so they reference the appropriate stackfile. In the SGI Irix systems, a user program can open up to 198 files. This number practically limits the total number of stackfiles that can be accessed at once.

2.2 Units and Resolution

Subroutine calls return 4-byte real values for height and other geophysical and environmental data such as the surface atmospheric pressure. 8-byte real values are returned for slot latitude and longitude. The units are degrees for latitude and longitude, meters for height and 10 Hz height displacement, millibars for pressure, 0.01 dB for Sigma-0, cm for SWH, and mm for other geophysical and environmental corrections.

In the stackfiles, numeric values are stored as signed integers and logical flags as bits to save space. There are conversion factors associated with each stackfile for conversion of real quantities, such as heights, lat/longs, pressures, and others, to integers that are actually packed in the stackfile. These factors are set when each stackfile is first created. The basic rule applied in determining these factors is that the original data resolutions on GDRs be retained. In fact, this results in obvious values for most of them. The only conversion factor that requires some deliberation is that of pressure.

The height conversion factor is the number of storage units per meter. For example, to store heights in millimeters, the height conversion factor should be 1000. Each bin header contains a height bias. Individual slot heights are packed as unit-converted, signed two-byte integers displaced from this bias. The extreme displacements of any slot height are, therefore, +32767/ *heightFactor* and -32768/ *heightFactor* meters. Currently, the height conversion factor is 1000 so they are +32.767 and -32.768 meters from the bin bias. If a new point is packed whose converted height displacement from the bias exceeds the signed two-byte limit, over-range slot flag is set and a data flag of "data missing" value is returned when the slot height is accessed. The 10 Hz height displacement is packed in mm. Thus its conversion factor is 1000 also.

The latitude and longitude conversion factor is the number of storage units per degree. At present, latitude and longitude displacements are packed in micro-degrees with the lat/long conversion factor 1,000,000. These lat/long displacements are packed in three-byte integers after this unit conversion. The extreme displacements are thus $+2^{23}$-1 and -2^{23} micro-degrees, going up to almost ±8.4 degrees.

Pressures range from 950 to 1050 millibars or so. A rule of thumb is that a change in pressure of one millibar affects the ocean surface height to rise by 10 millimeters (inverted barometer response of the ocean surface). Pressure is packed after removing a bias, multiplying the difference by a conversion factor and finally rounding to an integer. The recommended pressure bias is 1000 millibars and the conversion factor is 200, thus assuring a resolution of about 0.005 millibar.

Currently no conversion is made to pack into stackfiles all the other data, such as wet tropospheric, ionospheric, and ocean tide corrections, with a zero bias.

2.3 Flags

Bin headers and slots contain one-byte spaces for eight flag bits. The stackfile software automatically converts such flag bits to an array of eight logical values, that is, .TRUE. or .FALSE. In Fortran codes, one can write statements such as these for bin header flags:

```
      LOGICAL binFlags(8)
      …
      CALL sfGetHeader (sf, row, col, binLat, binLong, binFlags, …)
      IF (binFlags(1)) THEN
c….. over land
      …
      ENDIF
```

Here are the bin header flags currently defined.

Bin header flags	.TRUE.	.FALSE.
1	Land	Water
2 (**OUT OF DATE**)	No lake bias applied	Special lake bias applied
3	Oceans (water bins only)	Lake (water bins only)
4-8 unused		

Note: The land/water flag in the bin header has been initialized by the CSR from a digital elevation model different from the one of the ocean depth in the bin header. Two might disagree near coastlines. The land/water flag has the priority.

In the original version of CSR stackfiles, the special lake bias accommodates lake surfaces which are not at sea level. The special height biases for individual lakes are different from local geoidal heights. However, no lake data are stored in current stackfiles any longer.

The slot flags have the following meaning:

Slot flags	.TRUE.	.FALSE.
1. earth surface (**OUT OF DATE**)	ice	No ice
2. ocean tide correction	applied	Not applied
3. inverted barometer correction	applied	Not applied
4. point edited by preprocessor	removed (bad)	accepted (good)
5. height missing	missing	present
6. wild data point (3.5σ criterion)	wild	OK
7. height or lat/long displacement out of range	over-range	OK
8. Topex/Poseidon only	Poseidon	Topex

Flag 5, "height missing" has the first precedence. Users should check it first.

All slot flags are set true as an initialization when a stackfile is created (height missing). **SfInsert** routine of the stackfile software described below determines values of slot 5 (height missing), 6 (wild data point), 7 (over-range), and 8 when a height is inserted to a slot. The first four flags are copied from the source of the height information at that time.

2.4 Missing Data

The stackfile software uses a special REAL*4 data flag value to indicate "data missing" that can be returned from a stackfile routine. A user program can use the value to test data retrieved from an opened stackfile.

```
      REAL *4 missingData
      …
c….. get a special data flag value used to indicate missing data
      CALL sfGetMissingData (missingData)
      …
      CALL sfGetHeader (sf, row, col, binLat, binLong, …)
      IF (binLat .EQ. missingData .OR. binLong .EQ. missingData) THEN
        …
      ENDIF
```

2.5 File Size

The size of the Master.bin file is just 116 bytes. The Master.txt file only contains textual information. It will probably not exceed a few KB. The Journal file grows each time a user program accesses a stackfile. Over time, it might grow to several 100 KB.

Stackfiles are huge; on the order of 500 MB. Here is how to calculate the size. The fixed size of each stackfile is determined when it is first created. First parameter that determines this size is the maximum number of repeat cycles to be stored. For Topex/Poseidon, the current stackfile has been created to hold up to 400 cycles or about 10.8 years of data. Each bin of a height stackfile comprises a 28-byte header and 400 slots, each of which takes 9 bytes. Thus, for Topex/Poseidon, each bin requires 28 + 9×400 = 3,628 bytes.

The total number of bins is the product of the number of orbital revolutions per cycle and that of bins per pass (one half of revolution). The number of bins per pass is approximately 1/2 of the orbital period in seconds because one point per second is stored. To be symmetric across the equator, this number has to be an odd number. For the case of Topex/Poseidon stackfile, there are 127 revolutions in a repeat cycle and 3,141 bins per pass, or 127×3,141 = 398,907 bins in each of ascending and descending stackfiles. This amounts to 1,447,234,596 bytes, or about 1,447 MB for each of ascending and descending height stackfiles.

To maintain each data file's size no larger than about 100 MB, each of these height stackfiles are sectored. For current Topex/Poseidon stackfile, each of ascending and descending height stackfiles has been divided into 22 sector files. To accommodate 127 passes, each sector contains 6 passes that require about 68 MB to store up to 6×3,141 = 18,846 bins.

The same sectoring method applies to other stackfiles. Thus, each bin of 10 Hz - 1 Hz SSH displacement stackfile is 20×400 = 8,000 bytes long, for Topex/Poseidon, each displacement in a slot taking a 2-byte signed integer. Each sector file of 10 Hz SSH displacements requires 8K×18,846 = 150.768 MB.

Similarly, each bin of other stackfiles, such as those of surface pressure and corrections, is 2×400 = 800 bytes long, taking a 2-byte signed integer per slot. Therefore, each sector file of these has a fixed size of 15.0768 MB for the current Topex/Poseidon.

3 Fortran Calling Sequences

Routines that are used to modify and access a stackfile are described here. Users of these routines need not to worry about how a stackfile is implemented; how many sector files are used; the actual layout of data records, etc. These routines hide the unnecessary details of implementaion.

Note: The implicit convention of the Fortran INTEGER and REAL types is ignored here. For more controlled software development and maintenance, it is generally recommended to declare

```
IMPLICIT NONE
```

statement in Fortran codes.

3.1 Opening and Closing a Stackfile

CALL sfOpen (sf, name, access)

CALL sfClose (sf)

The **sfOpen** routine opens a stackfile for reading or updating. It returns a 4-byte integer value to **sf** argument. This value is similar to a logical unit number in conventional Fortran I/O. All stackfile routines that are invoked subsequently have **sf** as their first argument.

Name argument is a character string that is passed to **sfOpen** routine as the pathname of a stackfile directory to be opened. **Access** argument that is passed to **sfOpen** is a single character, 'r' for a session of read-only access and 'u' for updating access. The character has to be a lower case letter.

When the use of a stackfile is finished, **sfClose** is called. 4-byte integer argument **sf** that is passed to **sfClose** identifies the stackfile.

It is allowed to open several stackfiles at the same time and the value returned to **sf** argument serves for identifying different stackfiles. For example,

```
INTEGER *4 sfAsc, sfDes
CALL sfOpen (sfAsc, '/data2/sf/tp/ascending', 'r')
CALL sfOpen (sfDes, '/data2/sf/tp/descending', 'r')
…
CALL sfClose (sfAsc)
CALL sfClose (sfDes)
END
```

The following companion routines are useful when you are going to access satellite information in Master.txt file or time tag information in Equator.table file:

CALL StackOpen (sf, name, access)

CALL StackClose (sf)

Because routine **StackOpen** calls **sfOpen** and routine **StackClose** does **sfClose**, it is not necessary to call **sfOpen**/**sfClose** pair once **StackOpen**/**StackClose** pair is used for a stackfile. The calling sequence of two open/close pairs is identical. It is suggested to prefer **StackOpen**/**StackClose** pair to **sfOpen**/**sfClose** pair for obvious reasons.

3.2 Determining the Dimensions of a Stackfile

CALL sfGetSize (sf, rowBound, columns, cycles, firstCyc, lastCyc)

Once open, call **sfGetSize** to determine the actual stackfile dimensions. 4-byte integer **sf** argument is the stackfile number that is passed to the routine to indicate which stackfile is in question. **SfGetSize** returns the row size to **rowBound**, the column size **columns**, cycle size **cycles**, the first cycle number available in the stackfile **firstCyc**, and the last **lastCyc** (all 4-byte integers). All arguments of **sfGetSize** are positive numbers. The values of three parameters, **rowBound**, **columns**, and **cycles**, have been set when the stackfile was first created.

In a user's code, it is necessary to make sure that long enough arrays are reserved.

```
INTEGER *4 Ncyc
PARAMETER (Ncyc = 400)
REAL *4 heights (Ncyc)
LOGICAL slotFlags (Ncyc)
INTEGER *4 sf, rowBound, colSize, cycleSize, cyc1, cyc2
INTEGER *4 row, col

CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
CALL sfGetSize (sf, rowBound, colSize, cycleSize, cyc1, cyc2)
IF (cycleSize .GT. Ncyc) THEN
  PRINT*, 'increase cycle size Ncyc to', cycleSize
  STOP
ENDIF

DO row = -rowBound, rowBound
  DO col =1, colSize
    CALL sfGetHeader (sf, row, col, …)
    CALL sfGetHeightsFlags (sf, cyc1, cyc2, heights, slotFlags)
    …
  ENDDO
ENDDO
```

3.3 Reading and Writing Bin Headers

CALL sfGetHeader (sf, row, col, lat, long, flags, bias, depth, count, mean, stddev)

CALL sfPutHeader (sf, row, col, lat, long, flags, bias, depth, count, mean, stddev)

Two routines **sfGetHeader** and **sfPutHeader** access only the header portions of each bin. They have identical calling sequence and differ only in the directions of data travel. **SfGetHeader** gets bin header data from a stackfile while **sfPutHeader** places that information into a stackfile. The stackfile has to have been opened for reading before **sfGetHeader** is called and for updating if **sfPutHeader** is to be used.

Both routines begin with three 4-byte integer arguments that are passed to them for stackfile identification and locating a bin. Other arguments convey the header information from or to the stackfile.

4-byte real **lat** and **long** arguments are the geographic latitude and longitude of the bin center in degrees. Latitude range from -90 to 90 and longitude from 0 to 360 excluding 360.

Logical **flags** argument is an array of eight flags. The definition of each header flag is described in the section Header, File Structure.

4-byte real **bias** argument is the height bias in meters. Slot heights are packed as displacements from this bias.

4-byte real **depth** argument is depth of the ocean in meters and is positive downward from the geoidal surface. The stored values were interpolated from a low-resolution map.

4-byte integer **count** argument is the number of valid data points stored in the bin. The value is updated whenever new height data is inserted into the stackfile using **sfInsert**. The upper bound of this argument is the cycle size, which can be returned from routine **sfGetSize**.

4-byte real **mean** and **stddev** arguments are the mean and standard deviation of the valid height data in the bin. Units are meter. These values are updated whenever new height data is inserted into the stackfile using **sfInsert**.

Here is an example code that reads one header, changes a flag, and rewrites it.

```
IMPLICIT NONE
INTEGER *4 sf, row, col, count
REAL *4 lat, long, bias, depth, mean, stddev
LOGICAL flags(8)
…
CALL sfOpen (sf, '/data2/sf/tp/ascending', 'u')

row = 0
col = 1
CALL sfGetHeader (sf, row, col, lat, long, flags, bias, depth, count, mean, stddev)
Flags(2) = .FALSE.
CALL sfPutHeader (sf, row, col, lat, long, flags, bias, depth, count, mean, stddev)
```

```
CALL sfClose (sf)
END
```

3.4 Reading Slot Data

Routines **sfGetHeightsFlags**, **sfGetLatsLongs**, **sfGetPressures**, **sfGetWettropo**, **sfGetOceantid**, **sfGetEmbias**, **sfGetSwh**, **sfGetSigma0**, **sfGetIonocorr**, and **sfGetDssh10** return items from slots of a bin. The bin has to have been accessed already using **sfGetHeader**.

CALL sfGetHeightsFlags (sf, firstSlot, lastSlot, heights, slotFlags)

CALL sfGetLatsLongs (sf, firstSlot, lastSlot, lats, longs)

CALL sfGetPressures (sf, firstSlot, lastSlot, press)

CALL sfGetWettropo (sf, firstSlot, lastSlot, wet)

CALL sfGetOceantid (sf, firstSlot, lastSlot, otide)

CALL sfGetEmbias (sf, firstSlot, lastSlot, emb)

CALL sfGetSwh (sf, firstSlot, lastSlot, swh)

CALL sfGetSigma0 (sf, firstSlot, lastSlot, sigma0)

CALL sfGetIonocorr (sf, firstSlot, lastSlot, iono)

CALL sfGetDssh10 (sf, firstSlot, lastSlot, dssh10)

4-byte integer **sf** argument that is passed to these routines indicates which stackfile is to be accessed. The 4-byte integer **firstSlot** and **lastSlot** arguments are passed to these routines also and specify the range of repeat cycles to be read.

4-byte real array **heights** is returned with the sea surface heights of specified slots in meters and has the size n where n is the maximum number of slots. Any of them can have the value of "data missing" data flag. The sea surface height is the height above a reference ellipsoid. Argument **slotFlags** is an 8×n logical matrix to which slot flags are returned.

8-byte real arrays **lats** and **longs** are returned with the geographic latitude and longitude of slots in degrees. Any of them can have the value of "data missing" data flag. **Longs** values returned by routine **sfGetLatsLongs** might be slightly less than 0 or slightly larger than 360 near the Greenwich meridian. The size of these arrays, like that of the following arrays, is n, the maximum number of slots.

4-byte real array **press** is returned with the surface atmospheric pressures of specified slots in millibars. Any of them can have the value of "data missing" data flag.

4-byte real array **wet** is returned with wet tropospheric correction values in mm. The sign convention is the opposite of that in GDRs. Any of **wet** values can be that of "data missing" data flag.

4-byte real array **otide** is returned with the ocean tide corrections in mm. Any of **otide** values can be that of "data missing" data flag.

4-byte real array **emb** is returned with EM bias correction values in mm. The sign convention is the opposite of that in GDRs. Any of **emb** values can be that of "data missing" data flag.

4-byte real array **swh** is returned with the SWH in cm. Any of **swh** values can be that of "data missing" data flag.

4-byte real array **sigma0** is returned with the sigma-0 in 0.01 dB. Any of **sigma0** values can be that of "data missing" data flag.

4-byte real array **iono** is returned with ionospheric correction values in mm. The sign convention is the opposite of that in GDRs. Any of **iono** values can be that of "data missing" data flag.

Argument **dssh10** is a 4-byte real 10×n matrix to which 10 Hz - 1 Hz SSH displacements are returned in meters where n is the maximum number of slots. Any of **dssh10** values can be that of "data missing" data flag.

The following example code reads each bin of the entire ascending stackfile. It can handle up to the first 400 repeat cycles, but accesses only cycles 10 through 271.

```
      IMPLICIT NONE

      INTEGER *4 Ncyc
      PARAMETER (Ncyc = 400)
      INTEGER *4 sf, row, col, cyc, rowBound, colSize, cycleSize, cyc1, cyc2, count
      REAL binLat, binLong, bias, depth, mean, stddev
      LOGICAL binFlags (8)
      REAL *4 heights(Ncyc), press(Ncyc), swh(Ncyc)
      REAL *8 lats(Ncyc), longs(Ncyc)
      LOGICAL flags(8, Ncyc)
      …
      CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
      CALL sfGetSize (sf, rowBound, colSize, cycleSize, cyc1, cyc2)
      IF (cycleSize .GT. Ncyc) THEN
        PRINT*, 'increase cycle size Ncyc to', cycleSize
        STOP
      ENDIF

      DO row = -rowBound, rowBound
        DO col = 1, colSize
          CALL sfGetHeader (sf, row, col, binLat, binLong, binFlags, bias, depth,
     c                      count, mean, stddev)
          CALL sfGetHeightsFlags (sf, 10, 271, heights, flags)
          CALL sfGetLatsLongs (sf, 10, 271, lats, longs)
          CALL sfGetPressures (sf, 10, 271, press)
          CALL sfGetSwh (sf, 10, 271, swh)
```

```
      DO cyc = 10, 271
         …
      ENDDO
    ENDDO
  ENDDO
  CALL sfClose (sf)
  END
```

3.5 Getting the "Data Missing" Data Flag Value

When data have not been stored in a bin header or slot, a special REAL *4 value is returned by stackfile reading routines. The routine described here returns this data flag value:

CALL sfGetMissingData (missing)

4-byte real **missing** argument is set to the "data missing" value. It can be used to check data items retrieved from the stackfile.

```
      IMPLICIT NONE
      REAL *4 missing
      INTEGER sf, row, col, count
      REAL *4 binLat, binLong, bias, depth, mean, stddev
      LOGICAL binFlags(8)

      CALL sfGetMissingData (missing)
      CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
      row = 0
      col = 1
      CALL sfGetHeader (sf, row, col, binLat, binLong, binFlags, bias, depth, count,
     c                  mean, stddev)
      IF (binLat .EQ. missing .OR. binLong .EQ. missing)
     C   PRINT*, 'missing data in row, col:', row, col
      …
```

3.6 Logging

The stackfile software keeps a journal for each stackfile used. The journal is logged automatically each time a stackfile is created, opened, and closed. The logging information file is named as Journal and is kept in the same directory as other data files. Users can leave their own messages in the Journal file.

CALL sfJournal (sf, msg)

4-byte integer **sf** argument is passed to select a stackfile. A character string argument **msg** is passed to the routine and it appends a text message **msg** to the journal file whenever it is called.

Each call to **sfJournal** appends a single line to the journal file. The line includes the date and time, user id, name of the program, and a message text. A journal file is an ASCII file that can be viewed and printed.

```
INTEGER *4 sf
CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
CALL sfJournal (sf, 'starting loop A')
DO I = 1, 1000
…
ENDDO
CALL sfJournal (sf, 'all done')
CALL sfClose (sf)
```

3.7 Getting Text Stored in the Textual Master File

After a stackfile is open, routine **sfGetGItext** can be used to retrieve textual information given a group and item as look-up keys. This information is stored in Master.txt file in the directory of each stackfile.

CALL sfGetGItext (sf, group, item, text)

4-byte integer **sf** argument identifies a stackfile. 16-byte character **group** and **item** arguments are textual keys used for the retrieval. By convention, these keys are upper-case letters and trailing blanks in them are ignored. If a line with these keys is located, the remainder of the line is returned in character **text** argument.

Following example code retrieves the satellite name:

```
INTEGER *4 sf
CHARACTER *16 group, item
CHARACTER *20 satName
CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
group = 'IDENTIFICATION'
item = 'SATELLITE-NAME'
CALL sfGetGItext (sf, group, item, satName)
PRINT*, satName
```

Numeric values are also stored in this file. One can retrieve them with a code like this:

```
INTEGER *4 sf
CHARACTER *16 group, item
CHARACTER *100 text
REAL *8 period
CALL sfOpen (sf, '/data2/sf/tp/ascending', 'r')
group = 'EQUATOR'
```

```
item = 'PERIOD'
CALL sfGetGItext (sf, group, item, text)
READ (text(:25), *) period
```

In fact, this example is taken from a stackfile routine **get_stack_info**, which is invoked by **Stack_Open** routine. Both of these routines are coded in /data2/sf/tp/src/stacklib.f.

The numeric stackfile information in Master.txt file retrieved by routine **get_stack_info** is accessible through two Fortran COMMON blocks /equator/ and /stackinfo/. Users can use routine **get_stack_info** to access this stackfile information by including a header file /data2/sf/tp/src/stack_info.h in their codes.

3.8 Getting the Name of an Open Stackfile

A stackfile database consists of several data files under a stackfile directory. For example, there is a file containing equator-crossing data associated with every stackfile. To open such file, a program needs the pathname of the stackfile directory. This is what **sfGetName** routine returns.

CALL sfGetName (sf, name)

4-byte integer **sf** argument passed to this routine identifies a stackfile. **SfGetName** returns the pathname of the stackfile directory in the character argument **name**.

3.9 Avoiding Fortran Logical Unit Number Conflicts

The stackfile software uses **getlun** and **freelun** routines to get a free and release a recyclable logical unit number, respectively. Two routines maintain a pool of integer values, each of which is larger than 200 and used as a logical unit number of Fortran I/O. It is okay for users to use these two routines as well. In this way, a user code can be prevented from attempting to use a logical unit number that is being used for stackfiles.

CALL getlun (lun)

CALL freelun (lun)

When **getlun** is called, a 4-byte integer value is returned to **lun**. This value can be used in subsequent Fortran file I/O statements. When finished, one can call **freelun** to recycle the logical unit number to the pool of free ones.

4 Compiling Programs

Source codes of the stackfile routines described in this write-up can be found in the following three files of f90 source code package:

/data2/sf/tp/src/sf.f
/data2/sf/tp/src/share.f
/data2/sf/tp/src/stacklib.f

Two more f90 packages for stackfile applications are available:

/data2/sf/tp/src/mssLib.f
/data2/sf/tp/src/gridLib.f

The compiled object codes of these f90 source files are in /data2/sf/tp/src/lib/:

/data2/sf/tp/src/lib/sf.o
/data2/sf/tp/src/lib/share.o
/data2/sf/tp/src/lib/stacklib.o
/data2/sf/tp/src/lib/mssLib.o
/data2/sf/tp/src/lib/gridLib.o

To save disk space, it is recommended to link to /data2/sf/tp/src/lib instead of copying these files to your own lib/ directory. Besides, users can access updated version of these files always. For example,

ln -s /data2/sf/tp/src/lib lib

It is recommended also to have f90 compiler optimize your code for faster run of your executable object code. For example, suppose your f90 source code is in a file name as app1.f. Here is the command line to compile it, assuming the link mentioned above has been made:

f90 -O2 app1.f lib/sf.o lib/share.o

The executable goes to a.out.

5 File Structure

A stackfile is a directory containing several database files having standard unix file names each beginning with a capital letter.

Master.bin file is a binary file and contains two records: one for stackfile dimensions and the other for conversion factors and biases.

Most part of stackfile data is stored in sector files named as SectorA, SectorB, etc., for height and lat/longs. Other data, for example pressure, are in different sector files named as SectorApressure, and so on. Each sector file is a direct-access binary file. All rows and cycles (slots) of a certain number of columns are stored in each sector file having a fixed file size. The last sector file might have unused columns.

An ASCII file Journal contains a log of each program run that used the stackfile software and database directory. Each line contains date, time, user id, program name, and a message text.

Master.txt file is an ASCII file containing satellite and orbit constants that are useful to user applications.

Equator.table file is an ASCII file containing the equator-crossing information needed for the bin addressing of each data point that is to be inserted and for a time tag computation.

These five sets of database files are required in any stackfile directories. Master.txt and Equator.table files are not produced by the stackfile software.

5.1 Bin, Stackfile Record

Sector files are binary files. Each data record is called a bin and has a 28-byte header and a number of 9-byte height slots. The maximum numbers of rows, columns, and slots per bin are set when the stackfile is created. The slot size of 10 Hz - 1 Hz height displacements is 20 bytes. Ten 2-byte array elements of 10 Hz height displacements are indexed in the increasing temporal order in every slot. The slot size of other data, such as the surface atmospheric pressure, is 2 bytes.

5.2 Header

Five header items do not change even when a slot is updated: geographic latitude and longitude of the bin center, height bias, ocean depth, and bin flags. When a slot is updated, count, mean, and standard deviation of valid, that is, not missing, heights in the bin are recalculated and points exceeding 3.5 σ criterion are flagged as wild.

Six header values are stored as REAL*4: bin latitude, bin longitude, height bias, ocean depth, and mean and standard deviation of heights.

The bin latitude and longitude are packed in degrees.

To minimize slot size, slot heights are packed as displacements from a height bias of the bin. The height bias in meters has been copied from a geoid model.

The ocean depth in meters is stored in each bin.

The count, mean, and standard deviation of valid, that is, not missing, slot heights are calculated and maintained in the bin header.

Eight bits are packed with bin flags. Currently, two flag bits are used: (1) bin center is over land or water; (2) bin is in an in-land lake or ocean when bin center is over water.

Bin Header and Height Slot

	Type(bytes)	Units	Remarks
Header	28		
Geodetic latitude	REAL*4	degree	-90 … +90
East longitude	REAL*4	degree	0 … 360
Height bias (geoid)	REAL*4	meter	
Ocean depth	REAL*4	meter	positive downward
Mean height	REAL*4	meter	
Standard deviation	REAL*4	meter	
# points	2		
Flags	2		
Slot	9		
Flags	1		
Latitude displacement	3	micro-degree	
Longitude displacement	3	micro-degree	
Height displacement	2	millimeter	

5.3 Slot

It is important to minimize the slot size. For Topex/Poseidon, all height slots take 1,436 MB of disk space. Thus, each of latitude and longitude displacements in micro-degrees is packed into a three-byte field. This enables **sfGetLatsLongs** routine to return lat/longs of each slot with the precision of an 8-byte real.

To reduce the byte size needed to pack the sea surface height (SSH), the bin header keeps a height bias and each slot stores a displacement from that bias. This height displacement is packed into a 2-byte integer in mm. To return the SSH above a reference ellipsoid in meters, the height displacement in a slot is scaled and added to the height bias by **sfGetHeightsFlags** routine.

One-byte disk storage is used in each slot to pack eight-bit flag information. Slot flags are defined as:

1. ice/no ice surface (**OUT OF DATE**)

2. ocean tide correction applied
3. inverted barometer correction applied
4. point edited by preprocessor
5. data missing/exist
6. wild data point from 3.5 σ criterion
7. displacement out of range
8. For Topex/Poseidon, altimeter indicator. For other satellites, not used.

Flag 5, "data missing" has the priority over others. Currently, flag 1 (ice) is not used.

10 Hz height displacements from 1 Hz SSH are packed without shifting in millimeters in an independent set of slot files that are sectored exactly the same way as height slot files are.

Pressure data are kept in another set of slot files. The sector file names are SectorApressure, etc. The pressure displacement in each slot is packed in a two-byte integer. The pressure data has been subtracted by a bias and the resulting displacement scaled to be stored.

Other data, which are maintained in still other sets of sector files as two-byte integers, are not shifted nor scaled from the original GDRs. However, EM bias, wet tropospheric, and ionospheric corrections on GDR have been changed in sign (mostly from a negative to a positive).

5.4 Master.bin File

Master.bin file contains primary constants of the entire stackfile: dimensions, bias factors, and scale conversion factors. It is a binary file and its contents are never changed once the stackfile is created.

5.5 Master.txt File

Master.txt file is an ASCII file and contains other stackfile constants.

Each line consists of three fields: group, item, and data text. Group and item are identifiers used for grouping in the file. Data text is the actual value of each constant. For example, a certain set of polynomial coefficients is saved in this file. The polynomial name is used as a group identifier; the individual coefficient names are used as item identifiers; the coefficient values are data texts. The maximum length of groups and items is 16 characters and that of data texts is 100 characters. The convention is that group and item identifiers are upper-case letters.

The lines do not have to be in any order. Given a pair of group and item, stackfile routine **sfGetGItext** simply opens this file, scans all lines sequentially until a line starting with such a pair is read, and returns the data text of the line.

5.6 Journal File

Journal file is a list of accumulated use logs. Each time **sfOpen** or **sfClose** is called, a line is automatically appended to this file. Application codes can call **sfJournal** to log their own messages.

Each line contains date and time in yymmddhhmmss format. Next, there is an 8-character user id. This is the login user id and is a blank if a batch job accesses the stackfile. Next 16-character field is the name of program. The remainder of the line is the message passed to **sfJournal**.

REFERENCES

Kruizinga, G.L.H., Validation and applications of satellite radar altimetry, Ph.D. dissertation, University of Texas at Austin, 1997.

The New Stackfiles, CSR technical memo, University of Texas at Austin, 1998.